DOCUMENT RESUME

ED 060 626 EM 009 632

AUTHOR	Bork, Alfred M.
TITLE	Science Teaching and Computer Languages.
INSTITUTION	California Univ., Irvine. Physics Computer Development Project.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	17 Aug 71
NOTE	31p.
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	*Computer Assisted Instruction; Computer Graphics; Computer Programs; Display Systems; Program Descriptions; Programed Instruction; Programing; *Programing Languages; Programing Problems; *Science Instruction
IDENTIFIERS	FORTRAN

ABSTRACT

Computer languages are analyzed and compared from the standpoint of the science teacher using computers in the classroom. Computers have three basic uses in teaching, to compute, to instruct, and to motivate; effective computer languages should be responsive to these three modes. Widely-used languages, including FORTRAN, ALGOL, PL/1, and APL, are compared. The decline of FORTRAN as the most widely used language is predicted. Various conversational forms of languages are compared, and criteria are set forward for terminal languages. These criteria include ease in learning, editing facilities, attitudes toward subroutines, dialog writing, string manipulating facilities, array and matrix capability and others. (RB)

Science Teaching and Computer Languages

Alfred M. Bork
Physics Computer Development Project
University of California
Irvine, California 92664

August 17, 1971

Recently computers have been used increasingly for teaching in science and in other areas. The teacher should be concerned about which computers and which languages he should use within his course for the greatest ease. Traditionally the decisions about computers available on campus have been made by other computer users, particularly researchers, and by computer theoreticians. But as computers become more and more widely used in the classroom, teachers should rightfully play some role in selecting of computers.

It is commonplace to say that computers come with different language facilities and computational power. Computational power is easier to measure, so it often is a major determinant in computer selection. But the user is more affected by total system performance, a combination of computer hardware and programming support, or software. My purpose is to consider computer languages, and the implementations of these languages, from the standpoint of the science teacher using computers within the class. First, I will comment briefly about the types of usage. Then I will comment on the pros and cons of currently available computer languages for the purposes of science classes, first

asingly for teaching in science be concerned about which use within his course for ecisions about computers ther computer users, partic- oreticians. But as computers classroom, teachers should of computers.

come with different language mputational power is easier rminant in computer selection. system performance, a combina- ng support, or software. uages, and the implementations of the science teacher using ill comment briefly about the the pros and cons of currently rposes of science classes, first in computational mode (both batch and interactive), then in dialog mode.

A personal element is inherent in these comments on computer languages, as the choice is to some extent a matter of taste, as with all teaching. However, certain languages do have some objective advantages and disadvantages. I shall omit many languages which, while perhaps desirable, are available only on a limited basis. All the computational languages discussed are widely available and are likely to be around in the next few years. However, not all are available on all computers. No dialog software is as widely available as the computational languages, so our discussion there is oriented toward the types of languages available.

The reader should understand that I am expressing a view about which computer languages might <u>best</u> be used. I do <u>not</u> suggest that one should do no computer work within science teaching if he cannot use the languages I favor. Even the least desirable languages for particular applications are often quite powerful and useful; for many problems it scarcely matters which language is used. Thus, although I will argue against FORTRAN and BASIC as languages of choice for most purposes in science teaching, I think they should be used within classes if, as often happens, they are the only languages available on a system.

Motivation for Computers in the Classroom

At least three factors motivate the use of computers within science classes. First, most successful science students will eventually use computers in their research. The computer is a vastly important research tool with great potentialities even outside science. But the computer can harm science if used improperly. Just as we teach students to use essential pieces of laboratory equipment, science courses are increasingly concerned with the early introduction of computers in a subject-matter context. The goal is to display to students the strengths and weaknesses of numerical and symbolic approaches, setting aside the purely analytic approach now common in most undergraduate courses. I call this factor the tool use of computers, as a computer becomes one tool the student acquires in his mathematical arsenal during his undergraduate and graduate preparation. John Kemeny, President of Dartmouth, has recently argued that the computer is so essential in everyone's education that colleges and universities which do not provide adequate student computer facilities should not be accredited.

The second use of computers in science classes is instructional. More and more the computer is found to be valuable in teaching science. This use is not necessarily in contrast to the tool use of computers, but it can be in some circumstances. When the computer is used as a computational device, we are close to the tool use, because the student is using the computer as a computer. But in tutorial use, the student need know little about the operational details of the computer.

rs in the Classroom

motivate the use of computers within science
successful science students will eventually
research. The computer is a vastly important
at potentialities even outside science. But
science if used improperly. Just as we teach
ial pieces of laboratory equipment, science
ly concerned with the early introduction of
-matter context. The goal is to display to
and weaknesses of numerical and symbolic approaches,
ly analytic approach now common in most under-
all this factor the tool use of computers,
one tool the student acquires in his mathematical
ergraduate and graduate preparation. John
artmouth, has recently argued that the computer
ryone's education that colleges and universities
dequate student computer facilities should not

uters in science classes is instructional.
uter is found to be valuable in teaching science.
arily in contrast to the tool use of computers,
circumstances. When the computer is used
ice, we are close to the tool use, because
he computer as a computer. But in tutorial
know little about the operational details

A third role that computers play in classes is motivational. Computers can stimulate student interest in the subject.

I assume science teachers are not primarily interested in preparing students to become computer experts, so their use of the computer is not motivated by such a desire.

Computational Batch Languages

When scientists consider preparing students for later use of computers (the tool aspect) they tend to gravitate toward FORTRAN. FORTRAN is by far the most commonly used language for scientific calculation. Practically all computers come with FORTRAN compilers, so it is more nearly an universal language than any other language available, (although FORTRAN does often vary significantly from machine to machine). Nevertheless, I believe that teaching FORTRAN, given other choices, is probably a mistake, even from the tool aspect.

In this situation one would have to be a sooth-sayer to predict the history of computer languages in the next few years. No area of contemporary endeavor is more dynamic and changing than that of computers, with continual growth in new machines and new languages.

Even for the scientist interested in computers solely as a calculational tool the environment is changing. In this changing environment I cannot envision FORTRAN as a long-term future language for scientific computation. It seems reasonable to predict a slow and steady decline in the importance of FORTRAN relative to other languages used for similar purposes.

FORTRAN was the first widely used formula-oriented language and had enormous success. It did more to ease the task of scientific computation than any single development in the computer field. Many scientists think of it as the only practical language. Nevertheless, FORTRAN is about 15 years old, and showing signs of age. It has gone

udents for later use of computers,
tate toward FORTRAN. FORTRAN is
ge for scientific calculation.
FORTRAN compilers, so it is more
y other language available,
gnificantly from machine to machine).
g FORTRAN, given other choices,
tool aspect.

be a sooth-sayer to predict the
next few years. No area of con-
nd changing than that of com-
machines and new languages.

computers solely as a calculational
In this changing environment I can-
future language for scientific com-
redict a slow and steady decline
ve to other languages used for

ormula-oriented language and had
ase the task of scientific
ment in the computer field. Many
practical language. Nevertheless,
showing signs of age. It has gone

through a series of elaborations, some not necessarily consistent with the original formulation of the language. Some of the original features were related to the structure of the IBM 704, the machine for which it was initially implemented. Thus, in many ways, FORTRAN is an old and creaky language. Further, it does not allow the user to take full advantage of all the facilities of a contemporary computer, much less the computer to be available in a few years.

What are the restrictions and limitations of FORTRAN? The branching controls, which allow the programmer to set up programming loops, the IF and computed GO TO statements in FORTRAN, are cruder in form and less powerful than those which exist in more recent languages such as ALGOL and PL/1. The standard input/output facilities in FORTRAN are inflexible; writing FORMAT statements is a chore for the beginning FORTRAN programmer, and sometimes even a bother to the experienced programmers. Evidence to support this is found in the large number of FORTRAN installations that have implemented format-free forms of input and output. Furthermore, FORTRAN considered large collections of numbers, arrays and matrices only as an afterthought; thus only individual numbers in arrays can be directly referenced. Much scientific computation is oriented toward collections of numbers and some newer languages have more elaborate and far-reaching ways of handling them, as we will note.

The ability to control what happens during error conditions was not built into FORTRAN; the interrupt system, used in most modern computers in error control, did not exist in the early days of FORTRAN,

so FORTRAN does not allow the user to decide in his program how to handle error conditions which generate interrupts. FORTRAN is weak in string manipulating facilities, because its original design contemplated only numerical calculations. Further, these facilities tend to be machine-dependent. While symbol manipulation represents only a small fraction of scientific computation, the use of on-line symbol manipulation is likely to increase; students should at least become aware of the possibilities. While symbol manipulation can be done with such FORTRAN based languages as FORMAC, nevertheless it is not an entirely natural operation in FORTRAN.

The fact that FORTRAN is now the most common language might be viewed as a sufficient reason for teaching it in science courses. I have tried to argue that it will not continue to be as widely used as it is today; when many present students are using computers in later research they will be using other languages. In general, the argument for teaching whatever exists today seems weak. If this has been done in the late 1950's, for example, students would have learned computing techniques and languages which would not have been useful in their later professional career; they would have stayed with desk calculators, or they would have worked in machine languages. The teacher must make reasonable projections about the state of the world when his students will be out of school.

The main current rivals to FORTRAN for scientific calculations are ALGOL and PL/1. Both languages are more rational than FORTRAN, primarily because of later design; they could profit from experience with FORTRAN. Thus, in both cases the branching statements are more

ide in his program how
interrupts. FORTRAN is
cause its original design
Further, these facilities
ol manipulation represents
tation, the use of on-line
; students should at least
symbol manipulation can
as FORMAC, nevertheless
s FORTRAN.

non language might be
g it in science courses. I
inue to be as widely used
s are using computers
r languages. In general,
today seems weak. If this
mple, students would have
s which would not have
areer; they would have
ld have worked in machine
ble projections about
will be out of school.

ientific calculations are
rational than FORTRAN, pri-
d profit from experience with
ng statements are more

natural and richer, and programs can assume a natural structure. Both allow a more rational structure of a program into blocks.

PL/1 has the additional advantage of powerful array and string processing facilities, not necessary in all scientific computation but often useful. Both PL/1 and ALGOL have limited availability compared to FORTRAN and are therefore in restricted use in scientific computations. However, the situation may soon be different.

ALGOL is not a new development; the initial work was in 1958 and 1960. A new ALGOL, from 1968, is now becoming available. Although ALGOL compilers exist on many computers, including IBM machines, it is still not a popular language in the United States, and one cannot see promise of its increase, although the new versions may change this situation.

PL/1 at the moment exists primarily on IBM computers, at a number of levels. Other manufacturers have PL/1 compilers in development, and wide interest is being shown in PL/1 by the computer industry. Although PL/1 has often been vigorously attacked, and certainly is not an "idea" language, it has considerable advantages over FORTRAN for many scientific computations. Current PL/1 implementations on the IBM 360 system are about equivalent in compiling and running speed to FORTRAN on the same machine. Debugging with PL/1 should be easier than with FORTRAN because of such built-in debugging facilities as the ON conditions, allowing the user to control error situations. The ability to handle collections of numbers, arrays and matrices, has already been mentioned. Furthermore, being a

recent language, PL/1 allows the user the full facilities of the computer in ways that are difficult with FORTRAN. Thus, a job can be divided into a number of sub-jobs, to be executed independently, perhaps even simultaneously, and storage of variables can be controlled by the programmer during the course of running the program. The PL/1 user also has more control over how his numbers are handled internally, sometimes useful in particularly sensitive calculations.

With this information it seems reasonable to assume that during the next few years PL/1 will be increasingly used for scientific computation, and the use of FORTRAN will probably decrease. Perhaps a new challenger to PL/1 will appear, such as ALGOL58 or APL, but at present it stands to succeed through lack of competition. Undoubtedly FORTRAN will continue in wide use for many years, because of the large current investment in it for scientific programs. Nevertheless, I believe its importance will be steadily decreasing.

Students should encounter the computer in their courses through languages that fully use modern computer facilities, so they avoid becoming tied down with older technologies. So what one might conclude about languages in two years may be quite different than those here. Nevertheless, I think that of the most widely available batch languages, PL/1 is the most sensible to teach students today. Again I stress that this is the ideal situation; if one has only FORTRAN available, teaching it holds great advantages over teaching nothing at all.

user the full facilities of the
ılt with FORTRAN. Thus, a job
ub-jobs, to be executed independently,
storage of variables can be controlled
rse of running the program. The
over how his numbers are handled
particularly sensitive calculations.

easonable to assume that during
increasingly used for scientific
RAN will probably decrease. Perhaps
pear, such as ALGOL58 or APL, but
through lack of competition. Undoubtedly
e for many years, because of the
or scientific programs. Nevertheless,
steadily decreasing.

mputer in their courses through
computer facilities, so they avoid
chnologies. So what one might con-
rs may be quite different than
nk that of the most widely available
t sensible to teach students today.
ideal situation; if one has <u>only</u>
olds great advantages over teaching

Some languages developed for conversational systems--BASIC, JOSS, and APL--are available.as batch languages on several computers, a trend which will probably continue. I shall discuss the applicability of these languages to student situations in the next section on time-sharing; remarks about the languages there will apply to a large extent to their use as batch languages.

Conversational Computational Languages

Computers can be used with students in systems where each student works at his own input/output device. Many observers feel that most computer work will eventually be done in conversational time-shared systems of this type, where the student can "converse" with the computer.

Almost all computational languages for batch use are available in conversational form. In addition some languages have been designed primarily for terminals. Furthermore, because conversational languages are less standardized, many dialects of a language are often in use. So the relative effectiveness of a terminal language in a classroom may be implementation dependent.

Five languages are widely available for terminal use. First, FORTRAN, already discussed as a batch language, is available from a number of time-sharing services, and on many time-sharing computers. The most widely used time-sharing service, General Electric, has had a variety of FORTRAN available for a long time, with many non-standard features (a source of possible difficulty in moving programs to other systems); recently the language has become more standardized. FORTRAN is also available in the IBM 360 RAX amd CALL/360 systems. FORTRAN is now a "standardized" language (to be distinguished from "defined" languages) and most recent terminal implementations reflect this standardization. The FORTRAN IV available for the XDS 940 from Tymshare, the Comshare XTRAN, and the Sigma 7 Extended FORTRAN in UTS appear to be particularly useful implementations. JOSS

anguages

dents in systems where each student
device. Many observers feel that
ally be done in conversational time-
where the student can "converse" with

ages for batch use are available in
ion some languages have been designed pri-
more, because conversational languages
alects of a language are often in
eness of a terminal language in a class-
endent.

ilable for terminal use. First, FORTRAN,
language, is available from a number of
many time-sharing computers. The
service, General Electric, has had
for a long time, with many non-standard
difficulty in moving programs to
language has become more standardized.
the IBM 360 RAX amd CALL/360 systems.
d" language (to be distinguished from
recent terminal implementations reflect
RTRAN IV available for the XDS 940
XTRAN, and the Sigma 7 Extended FORTRAN
rly useful implementations. JOSS

originated at the Rand Corporation, and has been implemented (with other names) in many time-sharing systems. Some JOSS variants are CAL (XDS 940), PIL (360/50,67), ISIS (360/50), TELCOMP (various PDP machines), AID (PDP-10), etc. A JOSS program can identify individual parts and refer to these parts, facilities missing in BASIC, the next language discussed. Thus it leads more naturally to construction of subroutines (sub-programs) necessary for a complex problem than does BASIC. Most implementations of JOSS have also insisted on the existence of the immediate commands, single lines executed as soon as extended, often extremely useful in initial learning and in de-bugging, as will be indicated.

BASIC was first developed at Dartmouth College, and was taken by General Electric as the basis for its nationwide time-sharing effort. It is similar to JOSS in many ways. General Electric quickly became the major force in this market, and many later competitors felt that they had to offer similar service. Hence, BASIC is a widely implemented language, available on most time-sharing systems. BASIC was initially a very simple language designed for ease of use with unsophisticated beginning students. Gradually, as with most languages, it has been extended; these additions have differed widely from implementation to implementation. Most BASIC implementations use compilers and do not have immediate commands available, a fact to be discussed later; exceptions are Tymshare SUPERBASIC, and SIGMA 7 BASIC. Many of the more advanced forms of BASIC have string manipulating facilities, but they differ considerably in detail; SUPERBASIC implements PL/1 string manipulation functions, while General Electric BASIC has very different facilities. BASIC usually allows subroutines

only in in-line coding; and it has instructions for accessing such coding from elsewhere in the program. Most BASIC facilities have simple but powerful matrix operations. Some BASIC implementations restrict variable names, but most users do not find this to be a problem.

As previously indicated, PL/1 was developed as a batch language by IBM. Several conversational versions are available. One rather full implementation (from Allen-Babcock) uses the name RUSH and, from IBM, a similar version is called CPS (Conversational Programming System). A more restricted subset, but one where simple programs run faster, is available under Call/360.

The final terminal language described here, APL, is an outgrowth of a book by K. E. Iverson, A Programming Language. It was initially an experimental system at the IBM Watson Research Center. Recently it has become widely available both for commerical use and for those who have 360's. Implementations are under development for many other systems, including XDS Sigma 7, Burroughs 5700, CDC 6600, and CDC 7600. APL has a large collection of symbolic functions. As compared with the other languages mentioned here, APL has extremely powerful built-in array and matrix manipulating facilities. Although the beginner can use a subset which resembles the other languages, APL has many operators for handling collections of numbers; thus it performs not only the standard matrix and vector operations, but it also has powerful generalizations of these. Thus the matrix "product" can involve a wide variety of pairs of binary operators. APL functions most efficiently when calculations are arranged to

tructions for accessing such
Most BASIC facilities have
Some BASIC implementations
s do not find this to be a

loped as a batch language by
re available. One rather
) uses the name RUSH and,
PS (Conversational Programming
one where simple programs

re, APL, is an outgrowth of
anguage. It was initially
Research Center. Recently
commerical use and for those
er development for many
rroughs 5700, CDC 6600, and
symbolic functions. As
ned here, APL has extremely
lating facilities. Although
bles the other languages,
ctions of numbers; thus it
vector operations, but
hese. Thus the matrix
airs of binary operators.
lations are arranged to

use these operators. It has both direct execution, and a function mode. These functions constitute a versatile subroutining facility. The user does not have the full control of storage the PL/1 user has, but he has some control over which variables are known to which pieces of the program, more than is available in other languages. The string processing facilities are relatively elaborate, and the language has a well worked-out philosophy of work spaces for system library and long-term student storage.

Terminal Language Criteria

What are the criteria that might help the teacher choose among the different calculational terminal language possibilities? Reasonable standards for such evaluation can be formulated, and we can consider the languages mentioned here with regard to these standards. The results are a function of both the language and the implementation. I shall refer to the languages as supplied with the machine or as generally available. Some deficiencies can be overcome by skillful programs, but most users will have to work chiefly with what is provided.

First, the language should be easily learned by the beginner. This is not simply a function of the language, but has to do in detail with how the language is taught. Those who learned computer languages by the older grammatical techniques are amazed to find how quickly students can learn today. A time-sharing environment, where the beginner can play in a structured way with the language at the terminals, provides a particularly rapid way for developing programming skill. Although some differences are discernible in ease of initial learning of the various languages here, as I will note, these differences are probably small. I would contend that within the environment of the science class most students can learn enough about any of the terminal languages discussed here to work elementary problems in about three hours at the terminals. It should be emphasized that the beginner need not, and in most cases should not, learn all features of a language before starting to use it.

e teacher choose among
guage possibilities? Reason-
e formulated, and we can
th regard to these standards.
language and the implementation.
ed with the machine or as
can be overcome by skillful
rk chiefly with what is

rned by the beginner. This
, but has to do in detail
who learned computer lan-
es are amazed to find how
e-sharing environment, where
y with the language at the
way for developing pro-
es are discernible in ease
ages here, as I will note,
would contend that within
t students can learn enough
ussed here to work elementary
minals. It should be emphasized
cases should not, learn
ng to use it.

Nevertheless, some advantages are inherent in learning one system rather than another. A language which has direct or immediate statements can be learned faster from the terminal than a language which runs only a complete program. In this mode individual statements, not full programs, can be executed immediately as soon as they are typed in; so the student can readily learn the effect of the statement. All forms of JOSS, SUPERBASIC, SIGMA 7 BASIC, some FORTRANS, and APL have such capabilities. On the other hand most varieties of BASIC, many FORTRAN and most varieties of PL/1, do not have immediate commands, and so are somewhat harder to learn.

FORTRAN also presents some additional problems for beginners because of its "unnatural" statements. Here I am referring particularly to the IF statement, which in its elementary form is not intelligible unless explained, and to FORMAT statements; most of the other languages provide simple methods of input and output.

Experience shows that APL presents two slight difficulties for the novice. First, it has a different precedence rule for operators than students are (perhaps) accustomed to from ordinary algebra: every operator operates on everything to its right; so "2-3-4", typed in, leads to the response "3". Similarly, a x b + c means a x (b + c) in APL. This deviation from usual precedence, valuable for the advanced user, because of the many operations in APL, can be controlled by parentheses, and the beginner should be urged to place many parentheses in his expressions. (This is good advice for all programming languages, preferable to teaching precedence rules.)

The second problem for beginners in APL concerns the branching statements, which are very powerful, but do not have the simple mnemonic form that JOSS and BASIC branching facilities have; a few minutes more are needed to teach elementary branching in APL functions. The right-pointing arrow is the basis of branching, but the place to branch to is computed. However, because of the array handling facilities, fewer branching statements are needed.

But these difficulties, both with FORTRAN and APL, are relatively minor. After using many languages with students, I believe that the few conveniences for the beginning learner in one or the other are relatively minor considerations in choosing a language. The differences in initial learning are often overexaggerated in the literature from the vendor; any languages can be quickly learned if one tackles a reasonable beginning subset. The <u>way</u> the language is introduced to the students is a greater factor; the traditional lecture approach, based on discussing the grammar of the language, is slower than learning directly at the terminal.

Editing facilities can ease the student's approach to a terminal. Many students do not type well, and a convenient editing system can circumvent great frustration over typing errors. Further, programs of any complexity seldom run when first written, so they must be de-bugged and corrected.

Terminal languages and systems vary enormously, from implementation to implementation, in editing facilities. At least three aspects of editing are important. Almost all terminal languages allow

rs in APL concerns the branching
erful, but do not have the simple
IC branching facilities have; a few
h elementary branching in APL functions.
e basis of branching, but the place to
r, because of the array handling facili-
ts are needed.

ith FORTRAN and APL, are relatively
ages with students, I believe that
eginning learner in one or the other
tions in choosing a language. The
g are often overexaggerated in the
y languages can be quickly learned if
ning subset. The way the language
is a greater factor; the traditional
cussing the grammar of the language,
ly at the terminal.

e student's approach to a terminal.
, and a convenient editing system
on over typing errors. Further,
dom run when first written, so they
ed.

vary enormously, from implementation
facilities. At least three aspects
ost all terminal languages allow

editing at the line level, replacing a line in a program with a new line, adding a line at any place in the program, and deleting a line. Although facilities for line editing are different for different systems, they are roughly similar. Further, facilities are similar for correcting errors on the line currently being typed, although these may not be present, and they can differ in convenience for the beginning user. These allow the cancelling of individual letters, or retyping of the line, if errors are noted before the line is completed.

However, not all languages allow editing within a particular line after it has been entered, particularly for lines already in the system. Powerful editors allow flexible modification of programs stored in the computer. The use of an editing system to change previously entered material is a personal matter, but the two systems I have found most convenient are the XDS 940 editing system, available on all languages in the 940, and the APL editing system. The 940 editor is based upon using many of the control characters. It takes some time to learn what the control characters do, but then one can quickly and easily make changes within lines, with a minimum of retyping. A full 940 editing language is often useful for systematically changing many statements. The APL editing system is based on putting slashes, for deletion, or a number, the number of spaces to be inserted, under the line to be edited, and then placing these inserted characters within a newly typed line supplied by the system. Spaces are provided, and the typing element is conveniently placed. A small change within a complex line is easy with such a system. But APL provides no fast way of making systematic

changes in many statements. Another useful editing system is the Dartmouth editing system, and the XDS SIGMA 7 BTM-UTS EDIT is also powerful and effective.

The user should be warned that many implementations of the languages mentioned here have no within-line editing. Particularly for programs where individual lines become very complex this lack can be a severe handicap. Some other systems have powerful but very difficult editing systems, almost impossible for use by anyone other than a dedicated professional. Only personal experience can indicate which editing system is most desirable for student use, but the question is important. The prospective purchaser or lessee should "experiment" in some detail with the machines he is considering so that he can form his own judgment with regard to editors.

The science student is a beginner at the terminal for only a brief period. He may then face a long career, both in school and out, of increasingly active computer use. The opposite side to the question of how quickly one learns a language is the question of whether the language affords an opportunity for student growth, in terms of his knowledge of and use of computers. Most languages are simple to get started in but after one has learned the elementary material, little else may be available. Some of the implementations of BASIC and JOSS, although not all, are like this. On the other hand some languages have a rich superstructure, not necessary for the beginning user, but available as he develops and can write more and more sophisticated programs. Both PL/1 and APL exemplify this richness, and FORTRAN can also be extended beyond the elementary level. I believe language

ther useful editing system is the
e XDS SIGMA 7 BTM-UTS EDIT is

any implementations of the languages
ne editing. Particularly for programs
ry complex this lack can be a severe
ve powerful but very difficult editing
se by anyone other than a dedicated
erience can indicate which editing
dent use, but the question is important.
see should "experiment" in some detail
ing so that he can form his own

r at the terminal for only a brief
career, both in school and out, of
. The opposite side to the question
uage is the question of whether the
for student growth, in terms of his
s. Most languages are simple to get
rned the elementary material, little
the implementations of BASIC and
this. On the other hand some languages
necessary for the beginning user,
can write more and more sophisticated
mplify this richness, and FORTRAN
elementary level. I believe language

should allow for, and even encourage, student growth, letting him make better and better use of the computer facility.

An issue related to growth is whether a student can emerge from a language he has already learned. Given the dynamic nature of the computer field, most languages will eventually die and be replaced by newer and more effective languages. Furthermore, as the student reaches more difficult problems he may want to use specialized languages adapted toward these problems. Hence, the question of how quickly he can make the transition from his first language to a new language is important. Existing languages are difficult to judge from this standpoint, but I have had several examples recently of good students, brought up on BASIC in high school, who have had psychological problems in switching from BASIC to a more powerful language. This experience may represent a chance occurrence, or it may be a phenomenon observable for all languages. As experience develops with languages, we should keep close contact with "changeability." A person can become accustomed to a language so that it is a "pacifier" to the user, a retreat in moments of crisis. Perhaps a multilanguage approach right from the beginning will prevent students from becoming too tied down to one language.

Terminal language usage will often be partially replaced by batch usage, perhaps through remote job entry systems, later in the student's career. So we should ask whether a language the student is using from the terminal will lead to successful batch usage. It is clear that the languages which exist in both forms, primarily FORTRAN and PL/1, have transitional capabilities. However, most of

the other languages here are not batch languages, except for small machines, and so would demand a change in framework in moving from terminal to batch. This factor, however, may become less important as research users shift to more terminal use, and as terminals are increasingly used for batch entry.

The student user also finds it very useful to have effective disk storage of his programs between sessions. (Paper tape or magnetic cassette can also be used for off-line storage.) The facilities for disk storage depend on the hardware and implementation rather than on the language; costs for disk storage vary widely. The availability of convenient and easy to use system library facilities also is important for class use, because the student can be relieved of the burdens of writing minor associated programs, or can be supplied some programs by the instructor. Library programs and usability differ widely from system to system. Protection features, which allow the instructor to control who has access to the files are also useful. A valuable feature is the automatic save in some systems; if you lose connection with the computer, possibly because of terminal or line failure, whatever you currently had available will be available the next time you contact the computer. So an entire session of work will not be wiped out by an accidental mishap. This facility exists in the RUSH implementation of PL/1 and in APL; it is a valuable user oriented facility which should be present in all time-sharing systems.

As already suggested, I believe that the language's attitude toward subroutines is important. The ability to conceptualize a large problem

ch languages, except for small
ge in framework in moving from
ever, may become less important
inal use, and as terminals
y.

useful to have effective disk
sions. (Paper tape or magnetic
ne storage.) The facilities
vare and implementation rather
storage vary widely. The
to use system library facilities
ause the student can be relieved
ociated programs, or can be sup-
or. Library programs and usability
. Protection features, which
has access to the files are
the automatic save in some
h the computer, possibly because
er you currently had available
contact the computer. So an
wiped out by an accidental mishap.
plementation of PL/1 and in
facility which should be present

t the language's attitude toward
ity to conceptualize a large problem

as a series of solvable subproblems, making little problems out of a big problem, is often a critical stage in the solution of any problem, computer or otherwise. A language which naturally pushes a student toward this point of view, in an early stage of his education, is desirable. BASIC is perhaps the weakest in this regard, as subroutines are possible only by in-line coding in most forms of BASIC in current use; this seems pedagogically unsatisfactory. The part-structure of JOSS is a type of subroutining facility, but it does not usually allow a convenient use of named subroutines stored in system libraries. For FORTRAN and PL/1, the ability to use subprograms, system or user-supplied, may be system dependent, but the facilities are usually present. APL has a general subroutining facility in its function approach, and the standard IBM system comes with a useful library of functions; the work spaces and system facilities of APL also allow an easy user-oriented access of this library.

Programs of any complexity almost never work when first entered. An important part of using the computer for the science student (or for anyone) is the process of debugging, finding and correcting programming errors. The terminal system is particularly useful in debugging because it allows immediate correcting and rerunning. In correcting code, the editing system, already mentioned, is of great importance. The system facilities can ease the task of finding and correcting errors.

When a program malfunctions, the first concern may be what has happened during the calculation. What statement is currently being executed? What are the current values of the variables? Particularly if no

printout at all has occurred, the user often finds it useful to determine the values of certain variables. Languages that provide immediate commands, commands executed right away, are convenient here. As soon as the program stops, the student can determine the values of critical variables. As previously mentioned, JOSS and APL, and some implementations of other languages, have direct or immediate commands. Languages which can only run whole programs do not offer a facility for determining the values of variables on the spur of the moment, an annoyance when things are not going well.

Several other system facilities are also useful in debugging. One such facility is the trace, the ability to require that each time a statement is executed, the value assigned is to be printed, and the ability to see which statements are executed in what order. While all of the languages discussed allow tracing through the insertion of temporary statements in the program, which can be removed after the program is running, only APL has a built-in tracing mechanism, allowing the user to specify which lines he wants traced. A similar facility is a stop or breakpoint facility, allowing the user to request that the computer pause after certain points. Again APL is the only language discussed here that usually has such a facility, although it may be available in some implementations of other languages.

The power o
considered,
is often pr
tion of col
of the oper
languages h
forms of BA
matrix hand
kind. APL
for collect
any other l
and serious
of its deve

ser often finds it useful to
ables. Languages that provide
d right away, are convenient
the student can determine the
reviously mentioned, JOSS and
ner languages, have direct or
can only run whole programs
ing the values of variables
ence when things are not going well.

also useful in debugging. One
lity to require that each time
ssigned is to be printed, and
are executed in what order.
allow tracing through the insertion
ram, which can be removed after
a built-in tracing mechanism,
lines he wants traced. A similar
cility, allowing the user to re-
certain points. Again APL is the
sually has such a facility, although
ntations of other languages.

The power of the language in handling matrices and arrays should be considered, particularly as the students become more advanced. It is often profitable to think of an operation as involving the manipulation of collections of numbers, rather than, as with FORTRAN, thinking of the operation as manipulating individual numbers. Many scientific languages have natural facilities for handling collections. Most forms of BASIC have a simple but effective collection of special matrix handling operations. PL/1 also has some facilities of this kind. APL has an extremely powerful and versatile set of operations for collections of numbers in many dimensions. It goes far beyond any other languages here, because the developer looked carefully and seriously at the question of matrices and arrays from the beginning of its development.

Conclusions--Computational Languages

Based on these criteria, I believe that APL and PL/1 are clearly superior as computational terminal languages for use with science students. JOSS is somewhat below these two, and BASIC and FORTRAN I regard as the least desirable languages for student use. Again, the reader should remember that some of these aspects depend on the implementation of the language.

ages

e that APL and PL/1 are clearly l languages for use with science these two, and BASIC and FORTRAN anguages for student use. Again, some of these aspects depend on the

Dialog Languages

A special problem arises with regard to computer languages for preparing material for student-computer interaction. The student in this mode interacts with a program already in the computer, and so has only indirect access to the facilities of the computer. Relatively little science material of this kind is available, but a sizable effort is being expended on work in this area, and more material is appearing.

Review articles list thirty or forty languages designed for dialog preparation. Most of these are in extremely limited use; some have never been used by anyone except their original developers! Few dialog languages could claim to have national use; one exception is the IBM Coursewriter, one of the first such languages. Little material is currently available in any of them; unfortunately, more energy has gone into developing languages than developing viable teaching sequences.

I suggest that the development of specialized languages for dialogs at this time is probably a mistake, and I feel that the science teacher employing computers in this computer-dialogue direction is wise to avoid tying himself down too closely with any of the existing languages. As indicated, most of these languages have extremely limited use at present. Furthermore, such material that has been developed is often presented in a form almost unusable outside the original environment. Thus the MIT relativity material is usable only on systems which have ELIZA, and estimates for converting to

another system indicate that this would be a sizable job. Further, many of these languages and approaches are based on a particular teaching strategy, and, as the teacher may not care to follow this approach, these impose a severe limitation; the teacher should retain control over the teaching process.

Although many specialized languages have been developed, it is possible to write student-computer interactive programs in existing general purpose languages. Languages already used include SUPERBASIC, APL, PL/1, FORTRAN, and SNOBOL languages. Most of these languages are flexible enough so that with only minor additions, usually in the form of a few subroutines or functions written by the user, one can handle much conventional dialog material. They do not prejudice the form of this material, but allow the user to pick his own teaching method. Furthermore, general purpose languages are much more widely available than specialized languages, so material written in them has a greater chance of being usable elsewhere. Instructors have a greater chance of being already familiar with general purpose languages. My own feeling is that the development of computer-based instruction languages has been premature, and that at present it is in most circumstances more reasonable to write such material in the general purpose languages.

Assembly languages are restricted to particular machines, but they also present interesting possibilities for dialog-type teaching material, particularly if the macro facilities of the language are fully exploited. A strong advantage is flexibility; it is easy to add new macros and subroutines to the system, so the programs can

ould be a sizable job. Further,
hes are based on a particular
ser may not care to follow this
itation; the teacher should
ocess.

have been developed, it is possible
ve programs in existing general
dy used include SUPERBASIC, APL,
. Most of these languages are
inor additions, usually in the
ons written by the user, one
material. They do not prejudice
w the user to pick his own
ral purpose languages are much
zed languages, so material written
ng usable elsewhere. Instructors
ady familiar with general purpose
the development of computer-based
ature, and that at present it
nable to write such material in

o particular machines, but they
ies for dialog-type teaching
facilities of the language are
ge is flexibility; it is easy to
he system, so the programs can

react to pedagogical needs. Further, the programmer can have access to <u>all</u> the facilities of the computer. He can for example imbed subroutines in higher level languages where desirable. Our work at Irvine, The Physics Computer Development Project, has followed this approach.

Given the situation of many existing languages possible for dialogs, and the limited availability of most of them, it is reasonable that work in this direction should strive for a form as independent as possible of particular programming languages. A flowchart showing the general structure of the program and the various possibilities should be maintained in all cases, and this information should be available for users of other systems. Such documentation can make a piece of teaching material usable in a variety of different places, with quite different kinds of machines and languages available, with only routine additional work to adapt it to local conditions. I believe it is a mistake to end CAI projects with the coding in a particular language as the only product and I strongly urge anyone writing dialogs to consider more communicable forms. The exact details of the flowchart are not critical; they may depend on the nature of the program; several types of flowcharts for the same program may increase the usefulness of the program.

Future Language Development--Graphics

Certain current language developments, related to terminal facilities, seem important enough from the standpoint of the science teacher to merit comment.

The languages of most importance for future computational use in science are those which offer limited conversational graphic capabilities. I am not referring to a full-scale graphic system, which often costs hundreds of thousands of dollars per terminal; rather I am referring to a visual conversational system like the Culler-Fried system at the University of California at Santa Barbara and the University of California at Los Angeles, and the BRAIN system at Harvard University. Both systems are based on terminals in the $10,000 range, and offer graphic capabilities for many areas of teaching. Both systems are based on storage oscilloscopes. Several sources offer similar terminals, and within the past year a dramatic reduction in cost has occurred. Graphic facilities are still more expensive than teletypes, but recently have become competitive with the better typewriter-like terminals. Further developments may allow these terminals to be less expensive; a number of interesting new types of graphic terminals are currently available.

A graphic terminal without graphic language capabilities is of little use. The systems mentioned above, Culler-Fried and Brain, are only to a minor extent hardware developments. Their primary strength is in software; they have implemented powerful conversational languages with effective graphic capabilities within the language. Such

lated to terminal facilities,
of the science teacher

ure computational use in
nversational graphic
ull-scale graphic system,
of dollars per terminal;
sational system like the
California at Santa Barbara
Angeles, and the BRAIN system
based on terminals in the
ities for many areas of
rage oscilloscopes. Several
hin the past year a dramatic
facilities are still more
ave become competitive with
urther developments may
ve; a number of interesting
ntly available.

age capabilities is of little
r-Fried and Brain, are
ents. Their primary strength
werful conversational languages
in the language. Such

extensions could be made in some of the existing terminal languages, including those mentioned in this paper.

The details would depend to some extent on the type of graphics to be implemented. Generally the two existing graphic languages mentioned allow natural handling of arrays so the student can often consider that he is working with functions. Of the languages mentioned here PL/1 and APL would probably lend themselves most readily to convenient graphic extensions in this sense, because of their array capabilities. It seems very likely that such facilities will soon be available in APL.

I feel that the teaching rewards in a graphic environment, perhaps with graphic input as well as graphic output, will be enormous, and that the development of such facilities will noticeably effect the teaching process in many areas.

	Direct Commands	Subroutines - User Supplied	Extensive System Subroutines	Array & Matrix Capability	Editing Facilities	String Manipulating Facilities	Initial Learning	Simple Branching
PL/1 - RUSH, CPS	No	Good	Yes	Good	Fair	Excellent	Easy	Yes
PL/1 - CALL 360	No	Fair			Fair		Easy	Yes
APL - 360	Yes	Good	Yes	Excellent	Excellent	Good	Reason-able	No
JOSS - FOCAL	Yes	Fair	No	Poor		No	Very Easy	Yes
JOSS- CAL	Yes	Fair		Poor		No	Very Easy	Yes
JOSS - PIL	Yes	Fair	No	Poor	Fair	Fair	Very Easy	Yes
BASIC - GE	No	Poor		Good	Good	Fair	Very Easy	Yes
BASIC - H.P.	No	Poor			Fair		Very Easy	Yes
BASIC - CALL/360	No	Poor			Fair		Very Easy	Yes
BASIC - Superbasic (Tymshare - 940)	Yes	Poor		Good	Excellent	Excellent	Very Easy	
BASIC - 940	No	Poor		Good	Excellent		Very Easy	Yes
FORTRAN - Tymshare			Yes	Poor	Excellent		Reason-able	No
FORTRAN - GE	No		Yes	Poor	Good		Reason-able	No
FORTRAN - RAX	No			Poor			Reason-able	No